Try Anything Once: A Case Study using NUCCOR

Rebecca Hartman-Baker
Scientific Computing Group
May 11, 2010

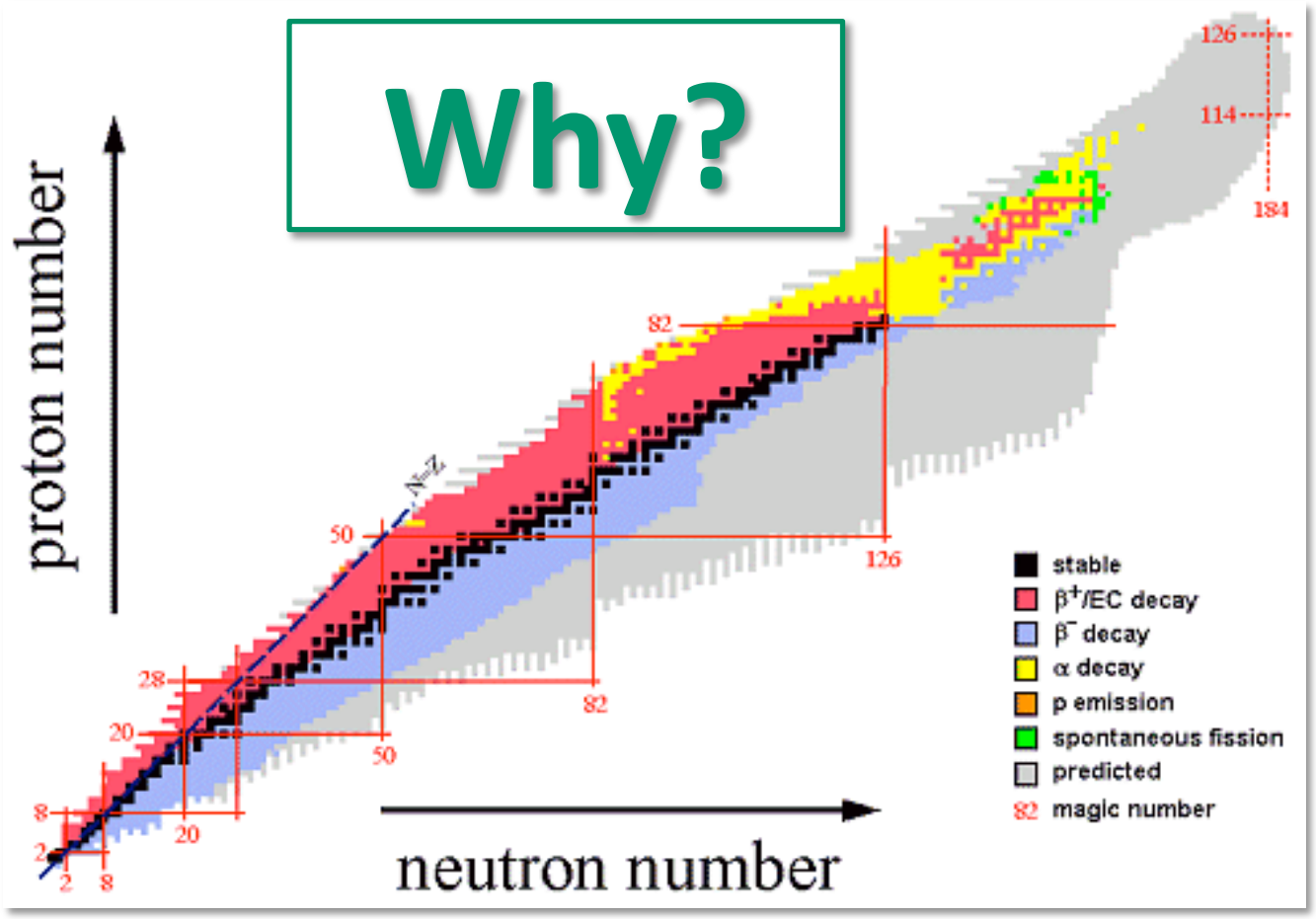
Outline

- Nuclear Physics in 5 slides or less
- Motivation for this work
- Experiments
- Results
- Addendum: Cray compiler and loop optimization
- Conclusions

Nuclear Physics: Basics

- Nucleus consists of protons and neutrons
- Like electrons, protons and neutrons in energy-level shells
- Some configurations more stable than others

Nuclear Physics: Table of Nuclides



Source: http://fys246.nuclear.lu.se/images/nucchrt2.gif

Nuclear Physics: NUCCOR code

- Nuclear Coupled-Cluster Oak Ridge code
- CCSD: approximates interactions between nucleons using single and double excitation states
- Essentially, solving huge set of coupled nonlinear equations
- Written in Fortran 90, with modules, derived datatypes
- Runs are constrained primarily by memory – lots of data, (relatively) low flop count

Motivation: My Job

- Improve code so that it
  - Scales better
  - Runs faster
- My (not-so-)secret techniques
  - Investigate/model performance of existing code
  - Implement better parallel I/O
  - Implement load balancing
  - Clean up wasteful/bad coding practices
  - Help code adhere to Fortran standards

Motivation: Events

- Grad student's version of code segfaulted only with Intel compiler, only with optimizations turned on, only on Jaguar
  - First guess: seat-to-keyboard interface problem
  - Turned out to be compiler bug!
- Talked to nuclear physicists
  - Depend heavily on Intel compiler
  - Reluctant to take risks with code
- I'm free to take risks and optimize code – I'm in this for performance, not science!
  - Make sure that optimizations do not result in incorrect answers, but nothing new comes out of my benchmarks

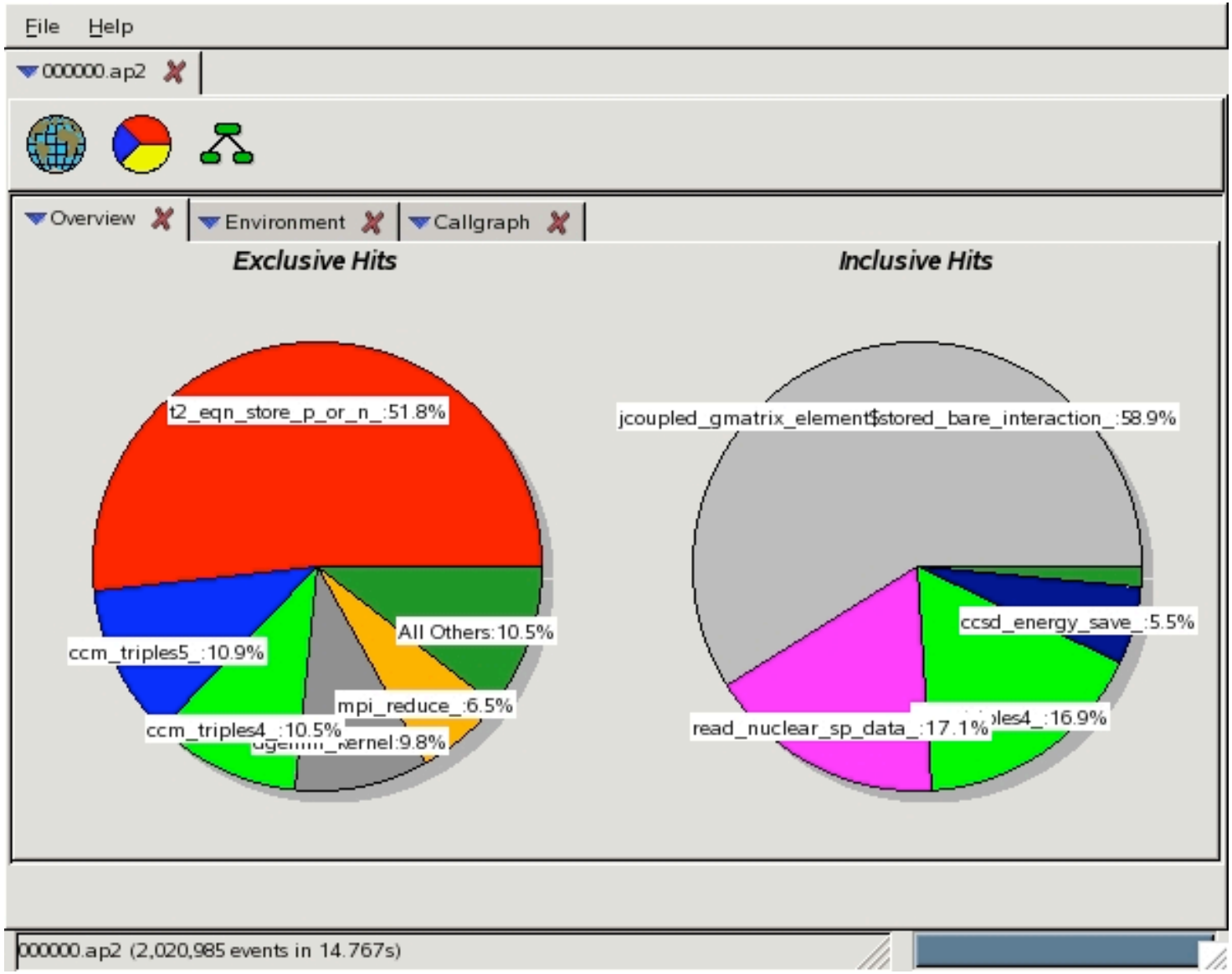
Motivation: Questions

- What is happening in this code?
  - Where is the most time spent?
  - What are the bottlenecks?
- How does the compiler impact performance?
  - Do compiler optimizations really matter?
  - Do different compilers perform differently?

Experiments: Performance

- Instrumented optimized code with Craypat
  - `module load xt-craypat apprentice2`
  - `make clean; make`
  - `pat_build -O apa mycode` (this creates executable called `mycode+pat`)
  - `qsub patscript`
- Generated reports using `pat_report` (this took a while)
- Viewed reports with Apprentice2: `app2 file.ap2` (loading this also took a while)

Craypat Results



Craypat Results

- Most time spent in `t2_eqn_store_p_or_n`
- This function full of deeply nested loops (very ugly)
- Text report provides info on which lines within subroutine take up the most time
- Many different pieces take up small portions of time, but overarching theme is time taken by performing iterative updates

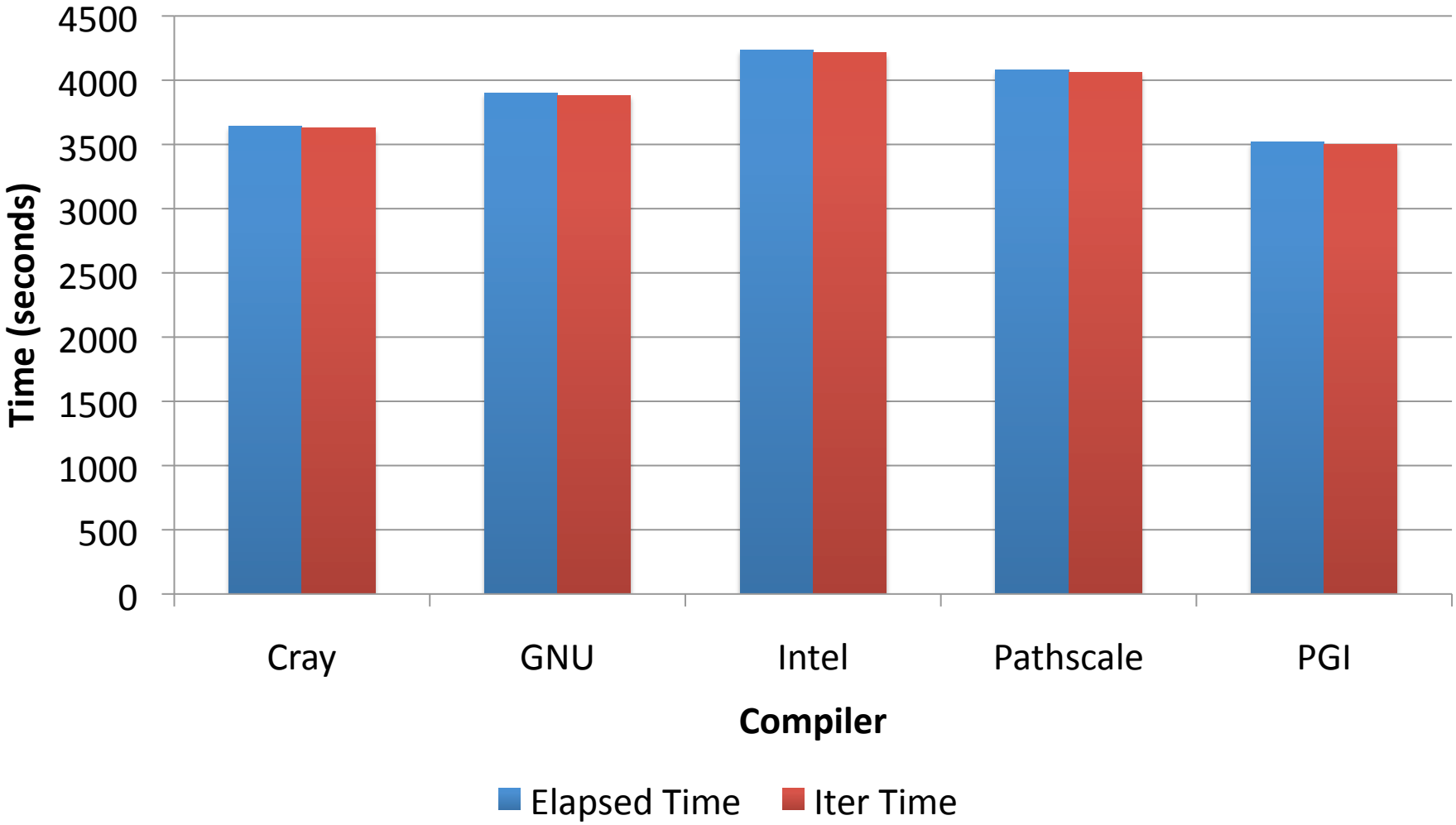
Experiments: Compiler Tests

- Compiled NUCCOR with all 5 compilers available on Jaguar (Cray, GNU, Intel, Pathscale, PGI)
- For each compiler, 6 different optimization levels (-O0, -O1, -O2, -O3, default, high)
- Ran each executable 3 times on ^{16}O benchmark, 441 processors, using 8 cores/node on Jaguarpf:
  - `aprun -n 441 -S4 nuccor.exe`
- Checked that each executable produced correct results (what good is getting the wrong answer quickly?)

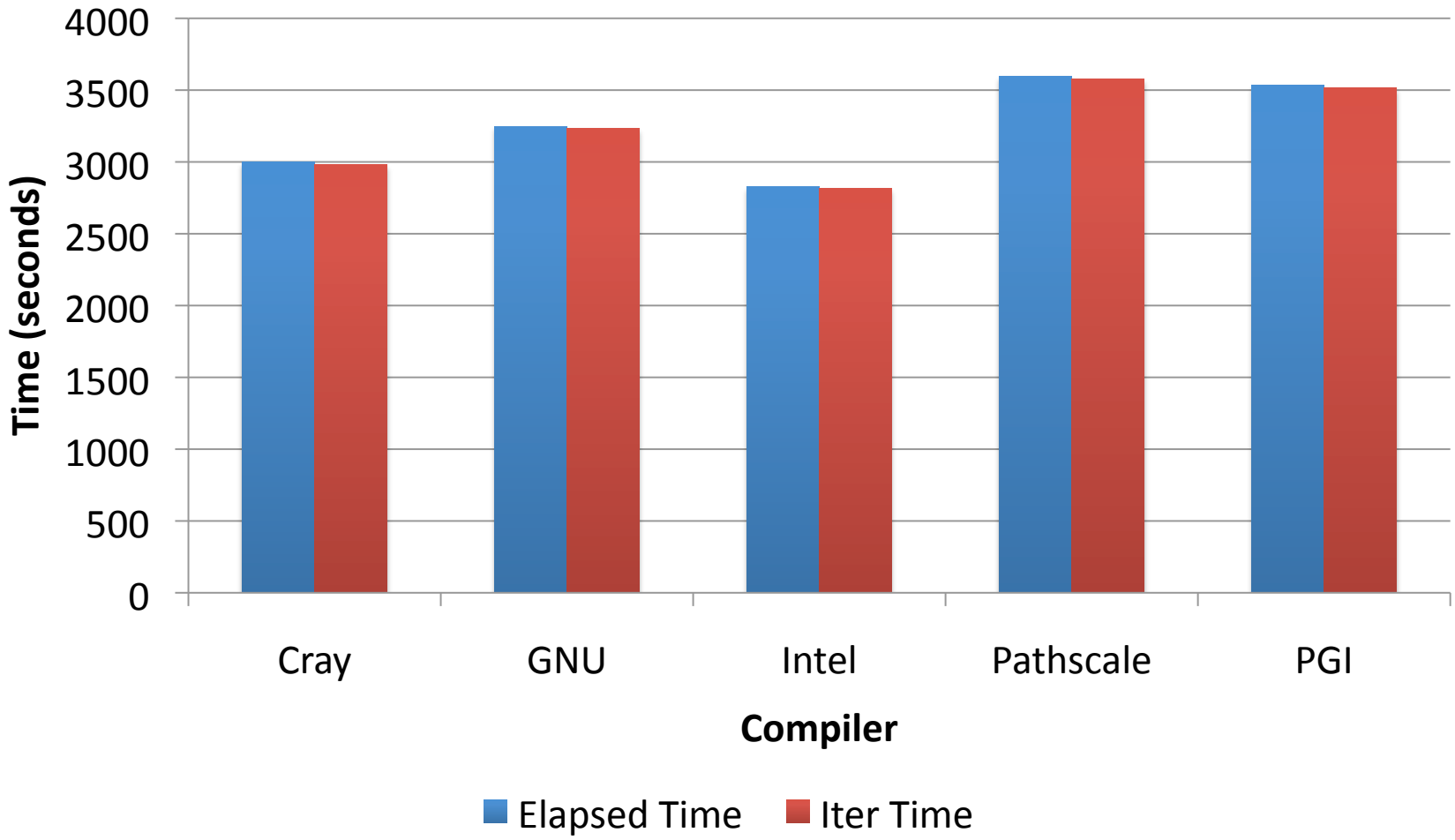
Experiments: Compiler Information

Compiler	Version	High Optimization flags
Cray	7.1.5 (default)	`-O3`
GNU	4.4.2 (default)	`-O2 -ffast-math -fomit-frame-pointer -mfpmath=sse`
Intel	11.1.046 (default)	`-O3`
Pathscale	3.2 (default)	`-Ofast`
PGI	9.0.4 (default)	`-fast`

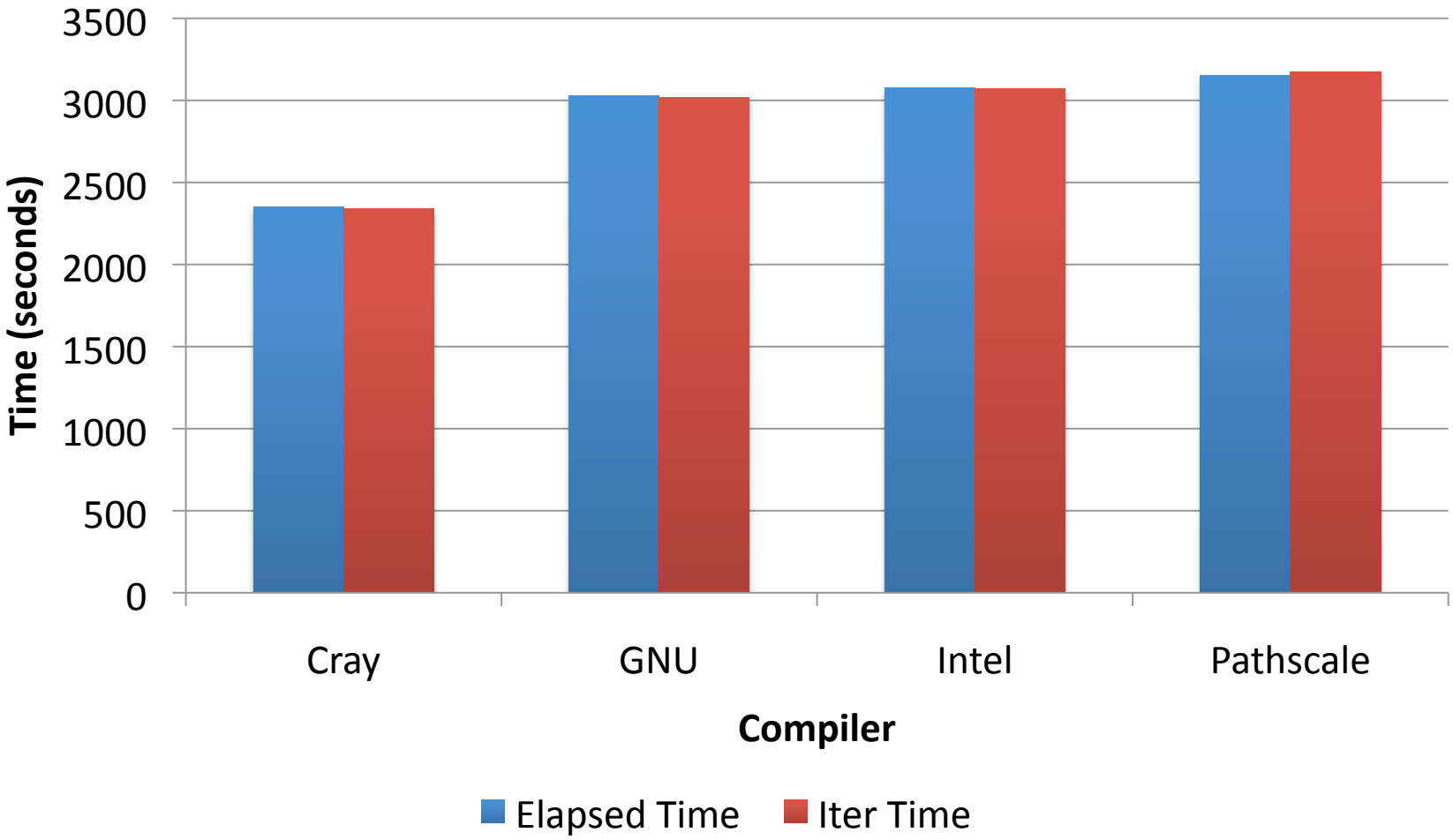
Results: -O0 Optimization Level



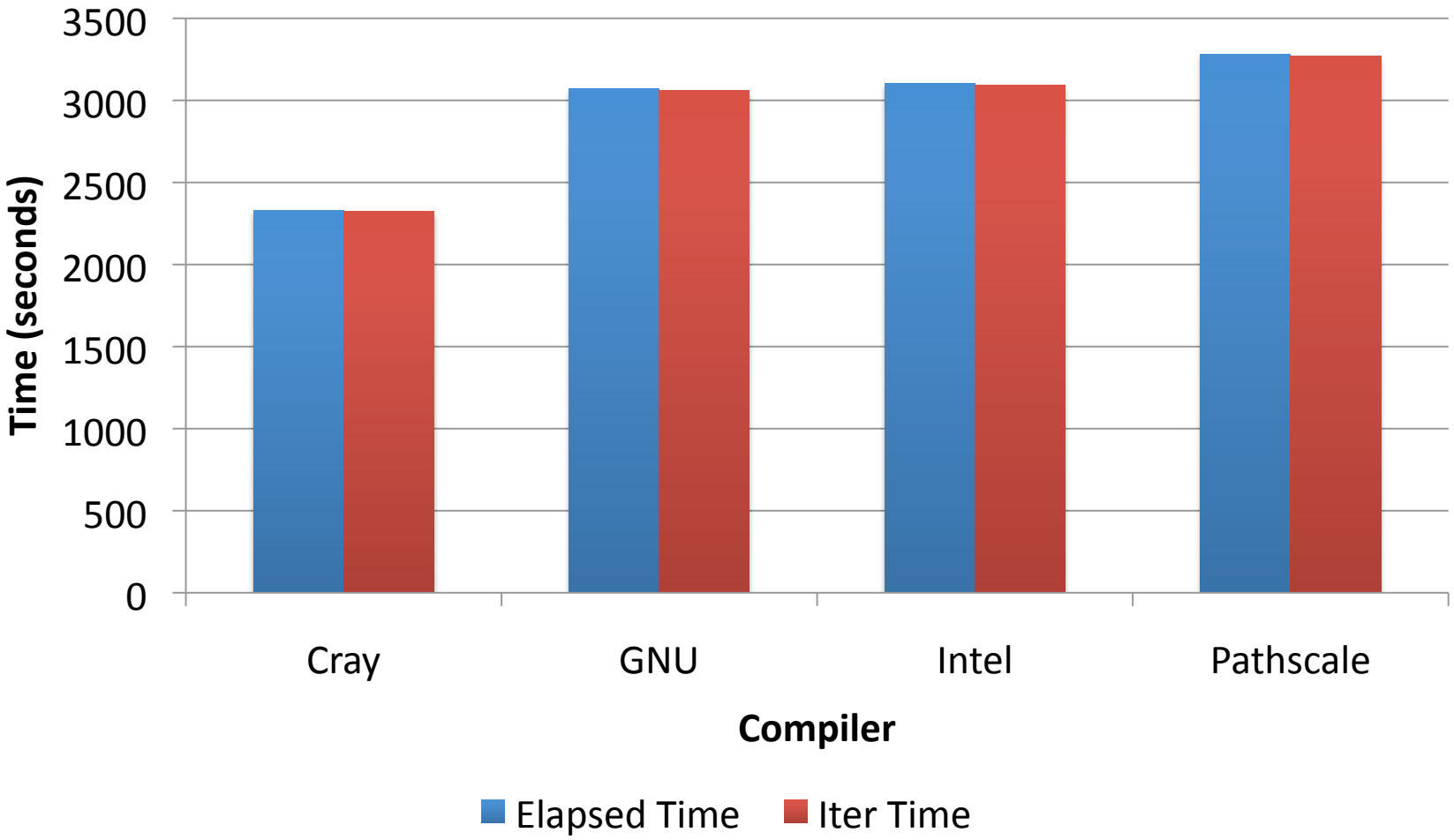
Results: -O1 Optimization Level



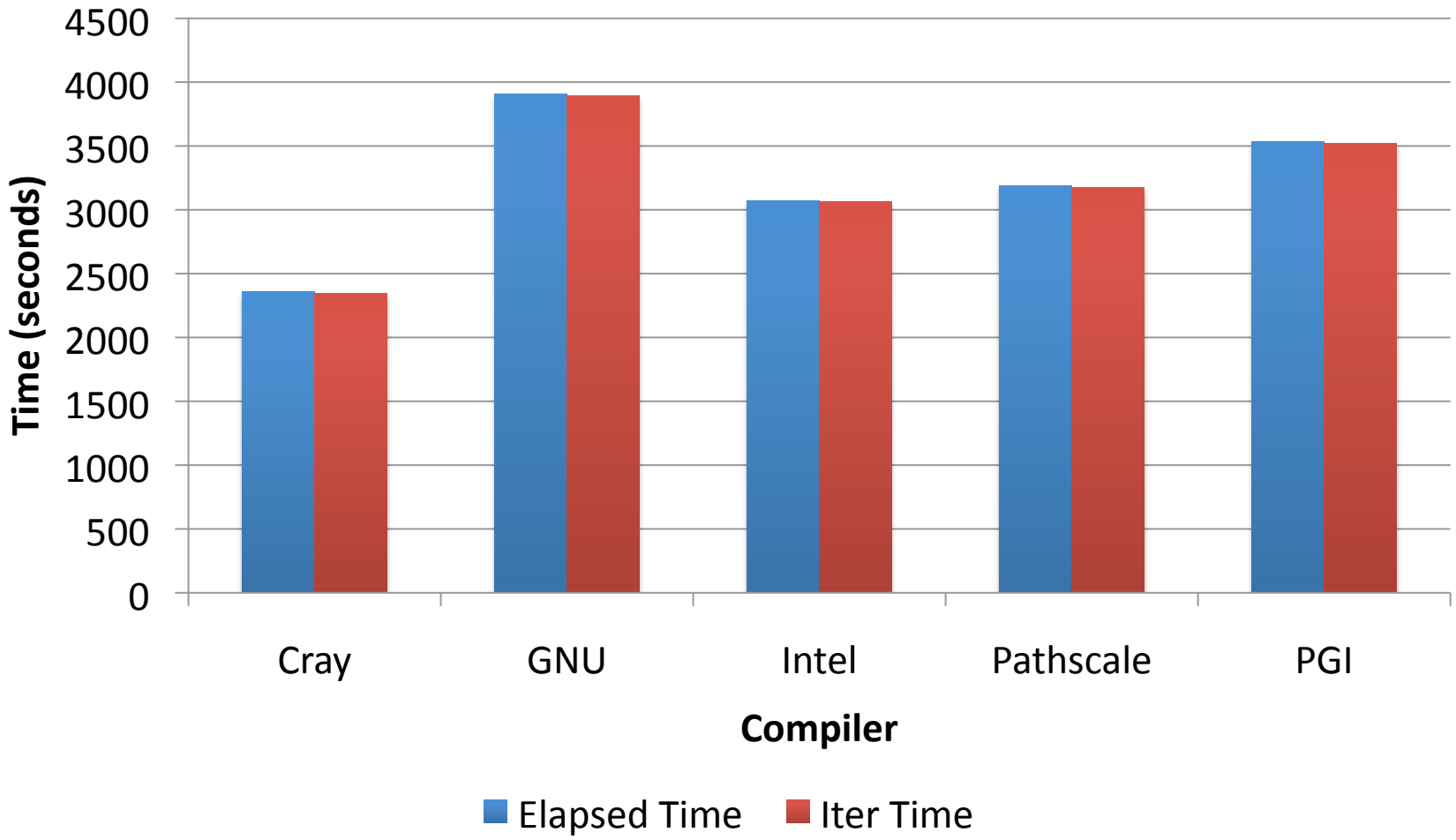
Results: -O2 Optimization Level



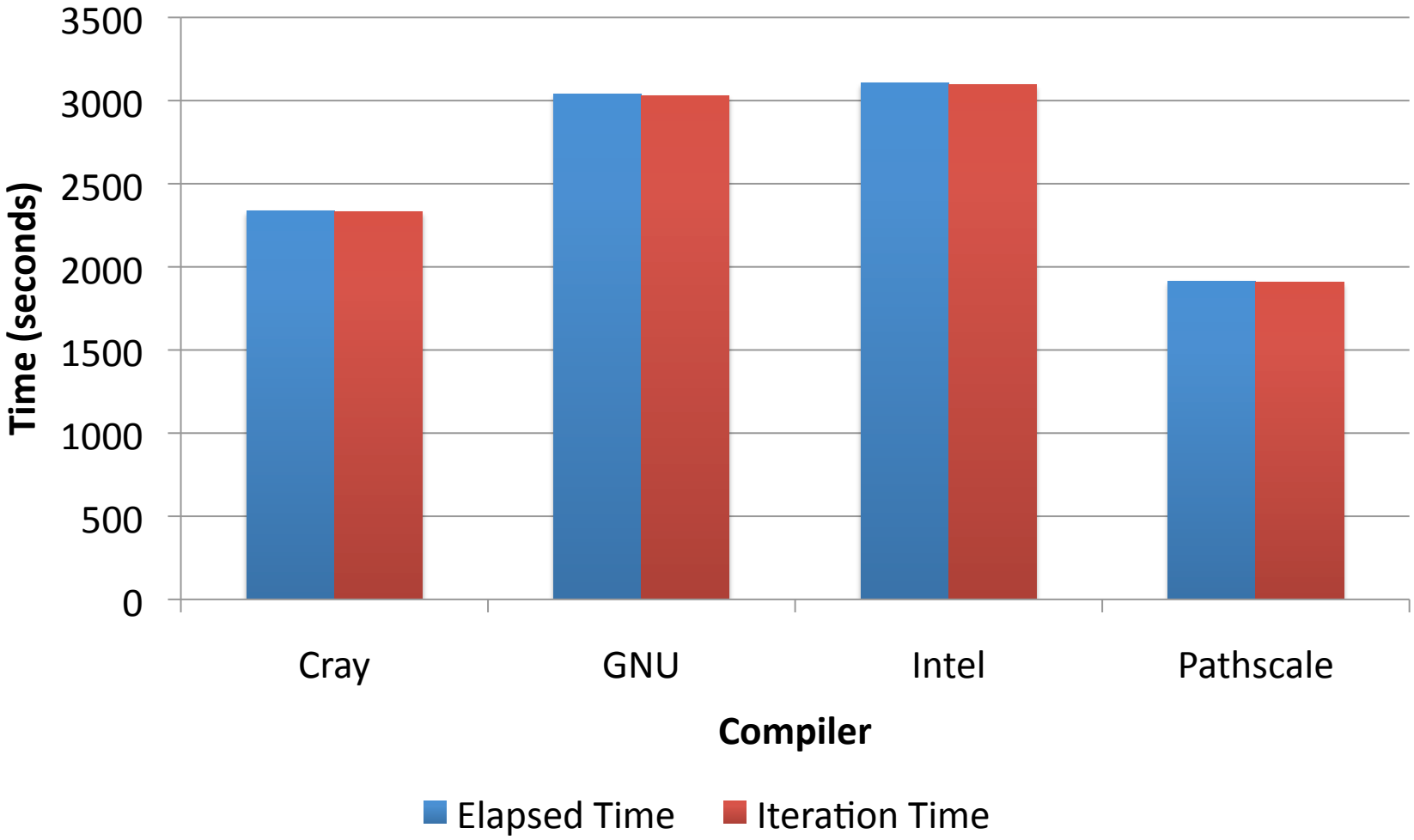
Results: -O3 Optimization Level



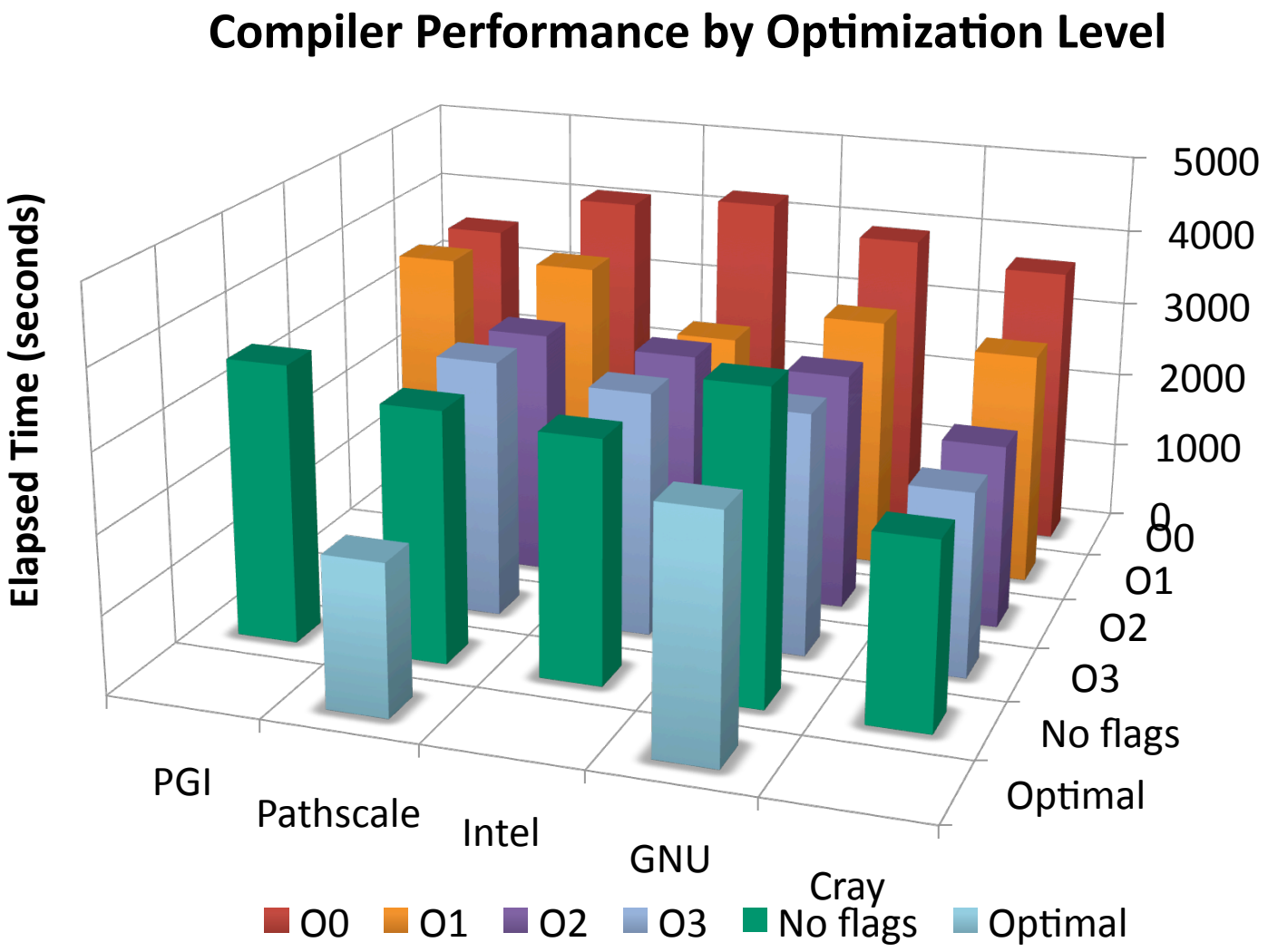
Results: Default Optimization Level



Results: High Optimization Level



Results: Aggregate Performance Results



Results: Best Performance at Each Optimization Level

Optimization Level	Top Performer	2nd Best Performer	% Difference in walltime between 1st and 2nd best performers
-O0	PGI	Cray	3.54
-O1	Intel	Cray	5.88
-O2	Cray	GNU	25.2
-O3	Cray	GNU	27.3
No flags	Cray	Intel	26.4
Optimal	Pathscale	Cray	20.5

Overall winner: Cray compiler!

Addendum: Cray Compiler

- I discussed results with Jeff Larkin, including surprise Pathscale victory
- He suggested sending code to Cray compiler developers, so they can improve their compiler
- Last week I received very nice, very detailed analysis of where Cray compiler did not optimize
  - Cray also opened ticket against this issue, and will fix it in next release
- Lessons can be applied to code and improve performance across all compilers

Addendum: Loop Optimization

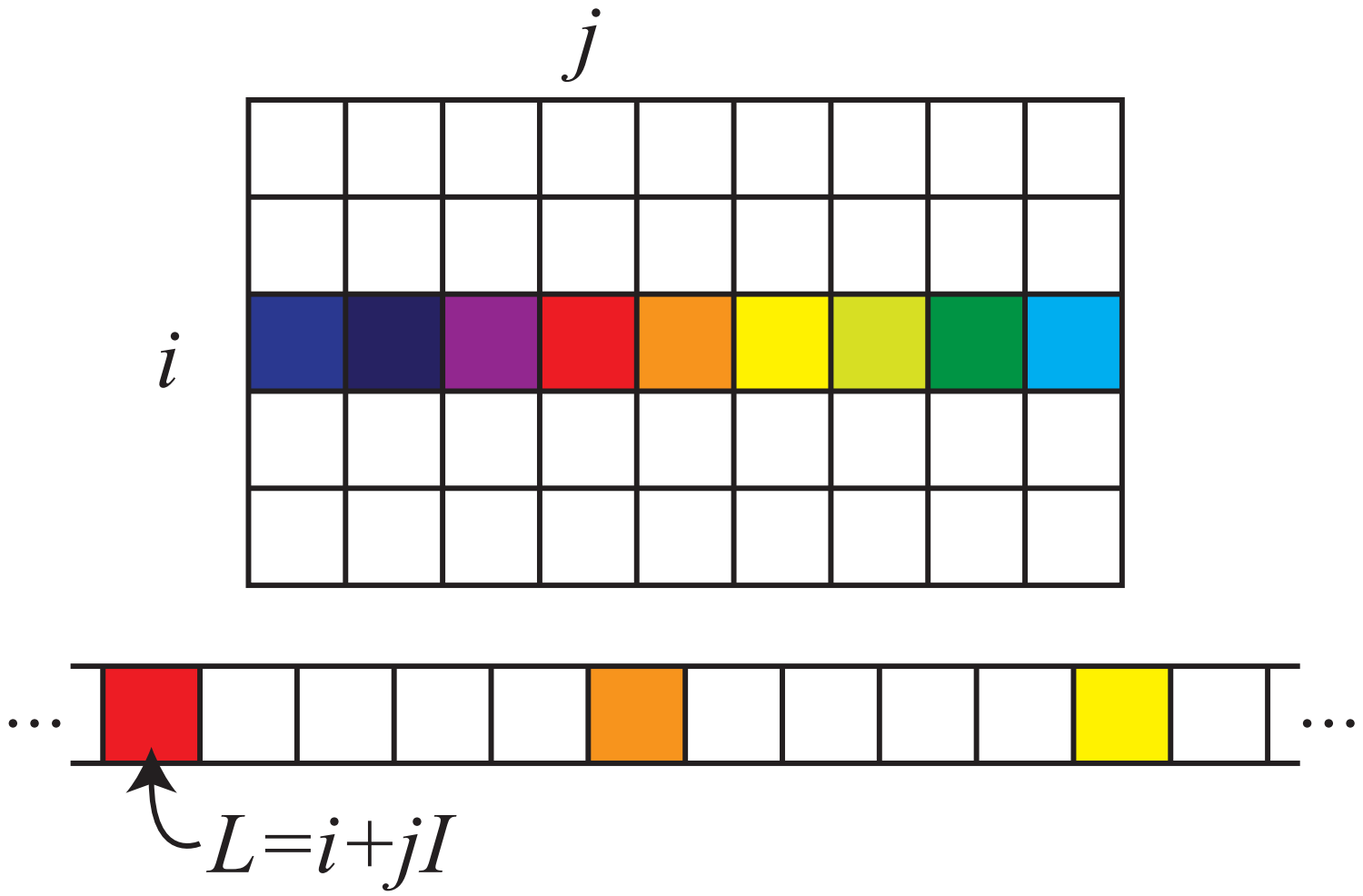
- Nuccor contains many deeply nested loops (depth 4)
- Loops written symmetrically for readability
- But, not easy for compiler to optimize

```
ii=0
do b=below_ef+1,tot_orbs
do j=1,below_ef
  ii=ii+1
  jj=0
  do a=below_ef+1,tot_orbs
  do i=1,below_ef
    jj=jj+1
    t2_ccm_eqn%f5d(a,b,i,j)= t2_ccm_eqn%
        f5d(a,b,i,j) + tmat7(ii,jj)
    t2_ccm_eqn%f5d(b,a,i,j)= t2_ccm_eqn%
        f5d(b,a,i,j) - tmat7(ii,jj)
    t2_ccm_eqn%f5d(a,b,j,i)= t2_ccm_eqn%
        f5d(a,b,j,i) - tmat7(ii,jj)
    t2_ccm_eqn%f5d(b,a,j,i)= t2_ccm_eqn%
        f5d(b,a,j,i) + tmat7(ii,jj)
    ops_cnt=ops_cnt+4
  end do
  end do
end do
end do
```

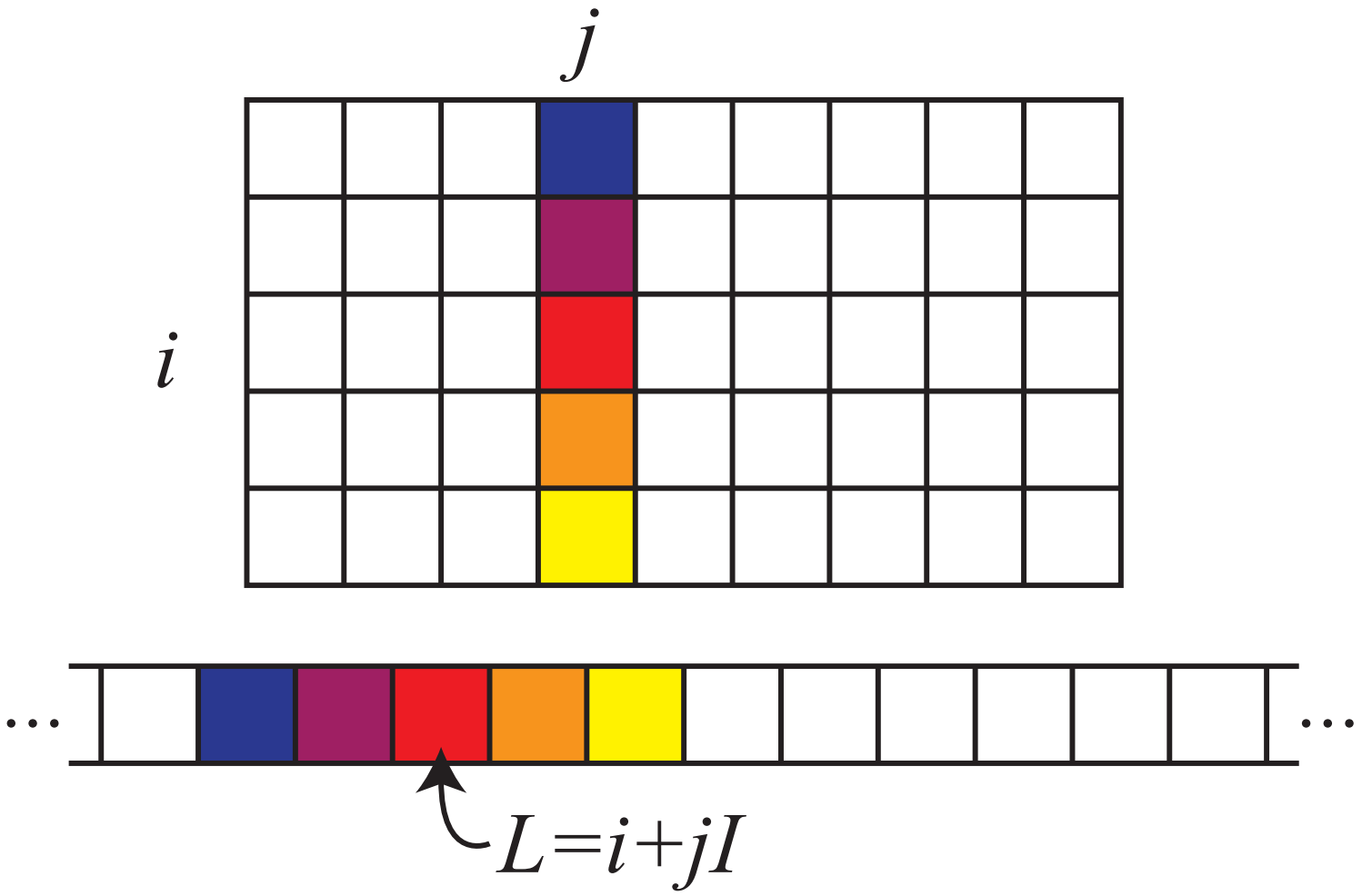
Problems in Loop

- Loop is easy for humans to read
- But, strides through memory cause cache thrashing and increased bandwidth use
- With `below_ef = 16` and `tot_orbs = 336`, each cache line of `t2_ccm_eqn%f5d` will have to be reloaded 8 times
- Also, array `tmat7(ii,jj)` referenced through 2nd subscript, so poor stride
- All compilers (except maybe Pathscale with `-Ofast`) fail to interchange loop nesting

Loop Memory Access (Poor Stride)



Loop Memory Access (Good Stride)



Compiler Optimizations

- Cray compiler will output annotated version of source file
  - `ftn -rm mycode.f90`
  - Outputs `mycode.lst`
- Examine annotated file to figure out what's going on

```
Primary Loop Type          Modifiers
------- ---- ----          ---------
A - Pattern matched        a - vector atomic memory operation
                           b - blocked
C - Collapsed              c - conditional and/or computed
D - Deleted                f - fused
E - Cloned
I - Inlined                i - interchanged
M - Multithreaded          m - partitioned
P - Parallel               p - partial
R - Redundant              r - unrolled
                           s - shortloop
V - Vectorized             t - array syntax temp used
                           w - unwound
```

Annotated Loop

```
371.                    ii=0
372.1--------<          do b=below_ef+1,tot_orbs
373.1 2------<          do j=1,below_ef
374.1 2                   ii=ii+1
375.1 2                   jj=0
376.1 2 3----<            do a=below_ef+1,tot_orbs
377.1 2 3 r8-<            do i=1,below_ef
378.1 2 3 r8                jj=jj+1
379.1 2 3 r8                t2_ccm_eqn%f5d(a,b,i,j)=… +tmat7(ii,jj)
380.1 2 3 r8                t2_ccm_eqn%f5d(b,a,i,j)=… -tmat7(ii,jj)
381.1 2 3 r8                t2_ccm_eqn%f5d(a,b,j,i)=… -tmat7(ii,jj)
382.1 2 3 r8                t2_ccm_eqn%f5d(b,a,j,i)=… +tmat7(ii,jj)
383.1 2 3 r8                ops_cnt=ops_cnt+4
384.1 2 3 r8->            end do
385.1 2 3---->            end do
386.1 2------>          end do
387.1-------->          end do
```

Unroll loop 8 times

Loop Reordering: Two Things to Try

- *Improve stride:* reorder so that `tmat7` is accessed by consecutive row, not column
- *Loop fission:* put all `f5d(a,b,:,:)` in one loop, all `f5d(b,a,:,:)` in another
- Test these two ideas in simple loop unrolling code

Test Code: Original Loop

```
ii = 0
do b = abmin, abmax
   do j = ijmin, ijmax
      ii = ii+1
      jj = 0
      do a = abmin, abmax
         do i = ijmin, ijmax
            jj = jj+1
            f5d(a,b,i,j) = f5d(a,b,i,j) + tmat7(ii,jj)
            f5d(b,a,i,j) = f5d(b,a,i,j) - tmat7(ii,jj)
            f5d(a,b,j,i) = f5d(a,b,j,i) - tmat7(ii,jj)
            f5d(b,a,j,i) = f5d(b,a,j,i) + tmat7(ii,jj)
         end do
      end do
   end do
end do
```

Test Code: Improved Stride

```
do i = ijmin, ijmax
   jj = 0
   do a = abmin, abmax
      do j=ijmin, ijmax
         jj = jj+1
         ii = 0
         do b = abmin, abmax
            ii = ii+1
            f5d(a,b,i,j) = f5d(a,b,i,j) + tmat7(ii,jj)
            f5d(b,a,i,j) = f5d(b,a,i,j) - tmat7(ii,jj)
            f5d(a,b,j,i) = f5d(a,b,j,i) - tmat7(ii,jj)
            f5d(b,a,j,i) = f5d(b,a,j,i) + tmat7(ii,jj)
         end do
      end do
   end do
end do
```

Test Code: Loop Fission

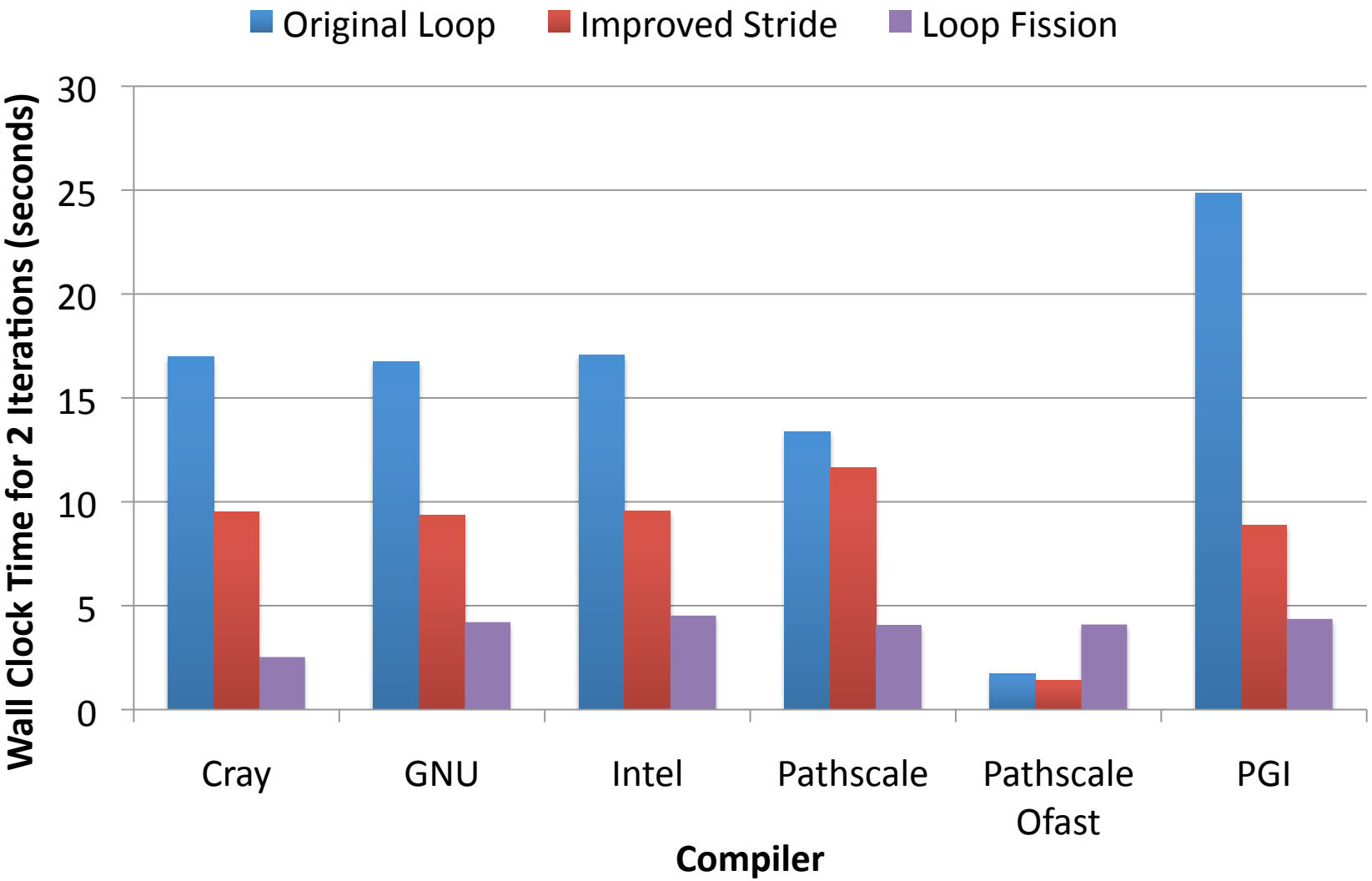
```
ii = 0
do j = ijmin, ijmax
   do b = abmin, abmax
      ii = ii+1
      jj = 0
      do i = ijmin, ijmax
         do a = abmin, abmax
            jj = jj+1
            f5d(a,b,i,j) =
              f5d(a,b,i,j) +
              tmat7(ii,jj)
            f5d(a,b,j,i) =
              f5d(a,b,j,i) -
              tmat7(ii,jj)
         end do
      end do
   end do
end do
```

```
jj = 0
do i = ijmin, ijmax
   do a = abmin, abmax
      jj = jj+1
      ii = 0
      do j = ijmin, ijmax
         do b = abmin, abmax
            ii = ii+1
            f5d(b,a,i,j) =
              f5d(b,a,i,j) -
              tmat7(ii,jj)
            f5d(b,a,j,i) =
              f5d(b,a,i,j) +
              tmat7(ii,jj)
         end do
      end do
   end do
end do
```

Test Code: Cray Compiler Behavior

- Original Loop: unrolled 8 times
- Improved Stride: conditionally vectorized, unrolled 2 times
- Loop Fission: 1st loop vectorized, partially unrolled 4 times; 2nd loop vectorized, unrolled 4 times

Test Code: Performance of All Compilers



Conclusions

- Things I learned from this exercise
  - Some parts of code were not standard Fortran (They are now!)
  - All optimizations produced identical results for this computation
  - Cray compiler is very good with Fortran, as advertised
  - Cray compiler developers very responsive to user feedback
  - Loop ordering makes HUGE difference
  - Try anything once!

Acknowledgments

- A very big thank-you to
  - Hai Ah Nam
  - David Pigg
  - Jeff Larkin
  - Nathan Wichmann
  - Vince Graziano
- This work was supported by the US Department of Energy under contract numbers DE-AC05-00OR22725 (Oak Ridge National Laboratory, managed by UT-Battelle, LLC).
- This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.